KYBERNETIKA ČÍSLO 5, ROČNÍK 3/1967

Realization of Generalized Good Translation Algorithm on Computer

JIŘÍ KOPŘIVA

In section 3, an algorithm performing so called *generalized good translation* of formal languages is given in the form of an ALGOL 60 procedure. Its description is given in section 2. It is possible to translate from a formal (context-free) language into other one with help of this algorithm if certain conditions hold for both grammars. At the same time, some parts of output text may correspond to certain chosen parts of input text in such a way that their relation is more general than the good translatability is.

1.

The notion of the *good translatability of grammars and languages* was introduced and exactly defined in references [1] and [2] and then generalized in [3]. The essence of the realization of good translation is certain transformation of the phrase marker of input sentence with the target to obtain the phrase marker of output sentence. A little modified form (comparing with [1], [2]) of the conditions of good and generalized good translation will be formulated here in order that an easier description of our algorithm may be given.

Let $G = (V_T, V_N, R)$ and $G' = (V'_T, V'_N, R')$ be *context-free grammars* with V_T and V'_T, V_N and V'_N, R and R' terminal alphabets, sets of intermediate symbols, sets of syntactic rules respectively. We write the rules $\mathfrak{r} \in R$ in the standard form

$$\mathfrak{r} = a_0 :: = b_k a_k \dots b_1 a_1 b_0 \tag{1}$$

with $a_0, a_1, \dots, a_k$ the nonterminal symbols and $b_0, b_1, \dots, b_k$ the (possibly empty) strings on V_T. Analogously for R'. The *language* L defined by grammar G is the set of all strings x on V_T such that there is a derivation (in the wonted sense used in the theory of context-free languages) with some $a \in V_N$ the first and x the last member. Analogously for L' and G'. Let S and S' be functions (*meanings*) defined on L and L' resp. The grammar G is said to be *well translatable* into G' if the following conditions hold:

a) *Let τ and Φ be mapping from V_N into V'_N and from R into R' respectively*

such that from

$$\mathfrak{r}' = \Phi(\mathfrak{r}) = c_0 : : = d_l c_l \ldots d_1 c_1 d_0 \quad (\mathfrak{r} \text{ is given in } (1))$$

follows $k = l$ *and* $c_{\pi(i)} = \tau(a_i)$ *for* $i = 0, 1, \ldots, k$, *where* π *is a permutation of the set* $\{1, \ldots, k\}$ (the permutation depends on $\mathfrak{r}$).

b) *If* $x_i \in L$ *and* $x'_i \in L'$ *is generable from* a_i *and* c_i *respectively and if* $S(x_i) = S'(x'_{\pi(i)})$ *for* $i = 1, \ldots, k$, *then*

$$S(b_k x_k \ldots b_1 x_1 b_0) = S'(d_k x'_k \ldots d_1 x'_1 d_0)$$

for all $\mathfrak{r} \in R$.

The *phrase marker* of a sentence of L (in the sense introduced in [1], [2]) *may be fully described with help of three arrays*, which are denoted *rule*, *sup*, and *dist* here. $rule_i$ is the *number of the rule labelling vertex* (*level*) i (the set R is supposed to be ordered; the order of m vertices in phase marker will be determined on analysing input sentence). In what follows, $rule_i$ means also *the rule* from R *itself*. sup_i denotes the *level labelled by the rule whose one intermediate symbol is replaced by the righthand side of* $rule_i$. At last, $dist_i$ is a *number labelling the oriented branch connecting vertices* sup_i *and* i. It determines the *distance* of the replaced symbol of $rule_{sup_i}$ from the end of its right-hand side.

The phrase marker of the corresponding sentence in L' will be obtained in the following way: *We replace the rules* $rule_i$ by Φ $(rule_i)$ *for* $i = 1, \ldots, m$, *and the numbers* $dist_i$ *by the corresponding numbers determined with help of permutation* π (for $rule_{sup_i}$) *for* $i = 2, \ldots, m$ (suppose the root of the phrase marker is level 1 with $dist_1 = 0$ e.g.). The mapping T from L into L' obtained in that way is called *good translation from language* L *into language* L'.

It is expedient (the reason is given in [3]) *to let some subtrees of the phrase marker in* L *avoid the good translation procedure. The left-hand sides of the rules labelling the roots of such subtrees belong to a set of chosen* (so called *auxiliary*) *intermediate symbols.* The part of input sentence corresponding to the (maximal) subtree of that property *may be translated or adjusted in an optional way* and then, during the synthesis of output sentence, *embeded on the relevant place* of it. Generalized good translation, described at length in [3], comes into being in this way. For it, the mapping τ need not be defined on the whole V_N.

2.

The elements of V_T and V_N are coded by numbers constituting arrays *term* (of length *tr*) and *nonterm* (of length *nt*) resp. The set R (from G) is coded with help of arrays *gram*1 (length *g*1) and *subj* (length *nr*). The right-hand sides of the rules from R are written sequentially and each of them is preceded by its negative ordinal number. The array *gram*1 ends then with the number $-z - 1$ if $z = \text{card } R$. Let us suppose that in the ordering of R *all rules having the same left-hand side* (so called *subject*) *form a segment.*

Array *subj* contains the subjects of all rules in the order given in R and ends with a zero.

In addition to the arrays *rule*, *sup*, *dist*, in description of the phrase marker, also arrays *init* and *final* are used. $init_i$ and $final_i$ denotes the subscript of the first and last element resp. of such part of input text (array *input* of length *il*), which is the value of the subject of $rule_i$. The elements of *input* are numbered from 1 to *il* from left to right. The element $input_0$, coded by the number *bsymb*, is placed on the beginning of input sentence so that the beginning can be recognized. The syntactical analysis is performed from right to left with help of an algorithm, which can be called *selective top-to-bottom right-to-left analysis algorithm.*

The reason for choosing a right-to-left algorithm is that it was tried out on a fragment of ALGOL 60 syntax containing also left-recursive but no right-recursive definitions. The top-to-bottom left-to-right analysis would yield no result in some cases.

On the basis of length *il* and of grammar, a predetermination of the upper bound *max* of the number of vertices of the phrase marker has to be done (here 3 – 4 times *il*).

The essence of analysis algorithm is rightmost derivation, but each met terminal symbol is compared at once with the corresponding input symbol. A kind of comparison is performed also in case a intermediate symbol of the rule is met. This is done with help of twodimensional boolean array *select* (with *nt* the number of lines and *tr* the number of rows). It is $select_{i,j}$ = **true** iff there exists a value of $nonterm_i$ with $term_j$ on its end and $select_{i,j}$ = **false** otherwise. Utilization of *select* on suitable places of algorithm precludes us from entering some false ways.

Syntactic analysis algorithm starts with labelling the first level by the first rule with subject *goal* (this is the code of the *main intermediate symbol*, the value of which the given sentence should be). On reaching the level n we label it by $rule_n$, which is the first rule having suitable subject. Then the symbols from the right-hand side of $rule_n$ are scanned and compared with the corresponding input text symbols. If some of them does not agree (in one or other way described in the foregoing paragraph), we return to the level m, where m is the maximal n reached up to now. We ask now whether there exists a next rule with the same subject as $rule_m$ has. This is done with help of array *subj*. If so, we replace $rule_m$ by this next rule; if not, we replace m by $m - 1$ (provided $m > 1$) and repeat asking about the existance of a next rule. If $m = 1$, the given sentence is not a value of goal.

If terminal symbol from the right-hand side of $rule_n$ agrees with the corresponding input symbol, we go ahead to the neighbouring left symbol in both the input and rule (the latter shift is realised in *gram*1). If this new symbol in *gram*1 is positive, we have not exhausted the whole $rule_n$ and we compare again. If this new symbol in *gram*1 is negative, $rule_n$ has been exhausted and, provided that $n > 1$, we return to the level sup_n (m remains unchanged) and determine the corresponding element in *gram*1. In case $n = 1$ we check whether $input_0 = bsymb$ has been reached. If not, we have a kind of disagreement and we return to the level m.

On meeting in *gram*1 a nonterminal symbol and if corresponding $select_{i,j}$ = **true,**

we increase m by 1 and put $n = m$. $rule_n$ is then the first rule with the met intermediate the subject.

Syntactic analysis algorithm is independent part of the whole algorithm. Corresponding procedure STBRL ANALYSIS is declared in the body of the whole translating procedure in such a way that on replacing formal parameters by actual ones one can use it independently of main algorithm. (See the note in the following paragraph.) It yields the phrase marker of input sentence in the form of output parameters (arrays) *sup, dist, rule, init, final.*

Function *scan* declared before analysis procedure is used also in other declared procedures and looks for the first occurence of u in array w of length v. Its subscript is *scan*; if there is no occurence of u in w, we put $scan = 0$. As a matter of fact, it should be declared in the bodies of the three main internal procedures.

Comments are put on the proper places of program so that the reader may be informed of what has been just performed. *Also following two procedures (translating and synthetizing ones) are written in such way that they may be used independently of main procedure.*

Procedure WELL TRANSLATE has arrays describing the phrase marker its input parameters. It transforms phrase marker in such a way that the result is a (*incomplete*, cf. [3], p. 310, IV) phrase marker of output sentence. Also arrays *subj* and *aux* (the latter of length a) belong to input parameters. The latter is a list of auxiliary intermediate symbols (cf. the end of sect. 1 and also [3]). Suppose $a \in V_N$ and $\tau(a) \in V_N'$ have the same numerical code (for all $a \in V_N$) and the numerical codes of terminals (in both V_T and V_T') differ from the codes of all intermediates, *only one specimen of arrays subj, nonterm and aux is sufficient for the whole program.*

The procedure we are just looking into, scans the vertices of phrase marker in the order determined by analysis. On each level n (beginning from the second), the value $dist_n$ is replaced by value corresponding to $\pi\,(dist_n)$ (permutation for $rule_{sup_n}$). This is done with help of arrays *pi*1 and *pi*2 (both of length *per*). The former has the form

$$(2) \qquad -q_1, q_{11}, \ldots, q_{1,r_1}, -q_2, q_{21}, \ldots, q_{2,r_2}, \ldots, -q_s, q_{s,1}, \ldots, q_{s,r_s}, 0\,,$$

where q_i for $i = 1, \ldots, s$ are numbers of all such rules of input grammar, which on the one hand have their images in output grammar and for which, on the other hand, some of its intermediates have changed their distance from the end of rule (in consequence of Φ). The original distances are $q_{i,j}$ in (2) and the resulting ones are the corresponding elements in *pi*2. The rest of *pi*2 is arbitrary.

Of course, if we meet $rule_n$ with an auxiliary intermediate the subject, then *after changing* $dist_n$ *we skip the whole maximal subtree with this vertex the root.*

Procedure SYNTHETIZE has also arrays *sup, dist, rule, init, final* its input parameters. In case array *aux* is empty the procedure synthetizes a sentence, the phrase marker of which is determined by the first three of the five mentioned arrays. The arrays *gram* (of length g and supposed to be coded in the same way as *gram*1 above) and *subj* are used for this purpose. The resulting sentence is then in array

output with $h + 1$ and -1 the subscript of the first and the last element resp. owing to synthetizing it from right to left.

Here, the considered procedure synthetizes the generalized good translation and the array *aux* is usually not empty. Owing to this fact, *the order of levels does not correspond to the rightmost derivation* on the one hand (values $dist_n$ have been changed), and *certain subtrees of the new phrase marker will not be used during synthesis* on the other hand. In order that a (optional) translation of those parts of input, which have the mentioned subtrees their phrase markers, to be embed into output, procedure TRANSL is declared. It translates sentence $A_B \ldots A_C$ in an optional way and thus forms sentence $D_E \ldots D_F$ and is used on a suitable place of the body of SYNTHETIZE.

It is obvious that array *gram*2 (of length *g*2), which is substituted for *gram* at the time of invokation of SYNTHETIZE *may contain only rules being images of such rules from gram*1, *which label vertices not belonging to the subtrees mentioned above. The corresponding rules are denoted by the same numbers.*

Realization of synthesis corresponds to a rightmost derivation. Therefore, *on meeting an intermediate, we have to find always the corresponding level.* Scanning e.g. $rule_n$ we have to find such level *d*, that *simultaneously* $sup_d = n$ *and* $dist_d$ *is the smallest one and not zero* ($dist_n$ has been put zero always after exhausting $rule_n$). Desired $dist_d$ is looked for as *min*, the value of which is $rl + 1$ at the very ouset and renewed, if necessary, after passing through a vertex. Here, *rl* is the length of the longest right-hand side from *gram*2.

I think, it is suitable to denote some formal and their corresponding actual parameters by the same identifiers.

3. THE ALGOL 60 PROCEDURE

```
procedure GENERALIZED GOOD TRANSLATION (gram1, g1, nonterm, nt,
          term, tr, subj, nr, select, goal, aux, a, input, il, bsymb, max, gram2,
          g2, rl, pi1, pi2, per, output);
value g1, nt, tr, nr, goal, a, il, bsymb, max, g2, rl, per;
integer g1, nt, tr, nr, goal, a, il, bsymb, max, g2, rl, per;
integer array gram1, nonterm, term, subj, aux, input, gram2, pi1, pi2, output;
boolean array select;
begin integer d, j, k, m, n; integer array sup, dist, rule, init, final [1 : max];
integer procedure scan (u, v, w); value u, v; integer u, v; integer array w;
begin scan := 0; for j := 1 step 1 until v do
                    if u = w[j] then begin scan := j; goto L end;
L: end scan;
```

```
424  procedure STBRL ANALYSIS (gram, g, nonterm, nt, term, tr, subj, nr, select,
         goal, input, il, sup, dist, rule, init, final);
     value g, nt, tr, nr, goal, il; integer g, nt, tr, nr, goal, il;
     integer array gram, nonterm, term, subj, input, sup, dist, rule, init, final;
     boolean array select;
     begin integer i; i := il; m := n := 1;
     if ¬ select [scan (goal, nt, nonterm), scan (input [i], tr, term)] then goto END;
     comment The last symbol of the input text is incorrect and so is the whole text. Thus
             run of program is stopped.;
     rule [1] := scan (goal, nr, subj); dist [1] := sup [1] := 0; final [1] := i; d := 1;
     L1 : k := scan (−rule [n] − 1, g, gram) − 1;
     comment Now, gram [k] is the symbol of the right-hand side of rule [n].;
     L2: if if scan (gram [k], nt, nonterm) = 0 then gram [k] ≠ input [i]
             else ¬ select [scan (gram [k], nt, nonterm), scan (input [i], tr, term)]
         then goto L4;
     comment Either the investigated symbol of rule [n] (in case it is terminal) differs
     from the corresponding input symbol or no values of the investigated symbol of
     rule [n] (in case it is intermediate) ends with the corresponding input symbol.;
     if scan (gram [k], nt, nonterm) ≠ 0 then
       begin
       m := m + 1; sup [m] := n; n := m; dist [n] := d; final [n] := i;
       rule [n] := scan (gram [k], nr, subj); d := 1; goto L1
       end: A new vertex is labelled by rule [n].;
     k := k − 1; i := i − 1; d := d + 1;
     comment: shift to the adjoining symbol in both the rule and input;
     L3: if gram [k] > 0 then goto L2;
                 if n ≠ 1 then
                   begin
                   d := dist [n] + 1; init [n] := i + 1; n := sup [n];
                   k := scan (−rule [n] − 1, g, gram) − d; goto L3
                   end;
     if input [i] = bsymb then
       begin init [n] := i + 1; goto L5 end: ascend to the higher
     level sup [n] after exhausting the rule or (for n = 1) the end
     of analysis;
```

```
L4: if subj [rule [m] + 1] = subj [rule [m]] then
      begin n := m; rule [n] := rule [n] + 1; i := final [n]; d := 1; goto L1 end
      descend to the last level m (maximal reached n) and shift to the following rule
      with the same subject (if any);
if m = 1 then goto END else begin m := m - 1; goto L4 end
incorrect input text (for m = 1) or decrease of the maximal reached level m if there
is no further rule being able to be used on the level m;
L5: end STBRL ANALYSIS;
procedure WELL TRANSLATE (subj, aux, a, pi1, pi2, per, sup, dist, rule);
value a, per; integer a, per; integer array subj, aux, pi1, pi2, sup, dist, rule;
for n := 2, n + 1 while n ≦ m do
    begin d := scan (-rule [sup [n]], per - 1, pi1);
      if d ≠ 0 then
      begin j := d;
      for j := j + 1 while pi1 [j] > 0 do
      if pi1 [j] = dist [n] then begin dist [n] := pi2 [j]; goto L1 end
      end dist [n] is given a new value obtained with help of permutation.;
    L1: if scan (subj [rule [n]], a, aux) = 0 then goto L2;
    for j := n, j + 1 while j ≦ m ∧ sup [j] ≧ n do d := j;
    n := d;
    comment The subtree with an auxiliary intermediate the root is skipped.;
    L2: end WELL TRANSLATE the phrase marker;
procedure SYNTHETIZE (gram, g, rl, nonterm, nt, subj, aux, a, output, sup, dist,
        rule, init, final);
value g, rl, nt, a; integer g, rl, nt, a;
integer array gram, nonterm, subj, aux, output, sup, dist, rule, init, final;
begin integer h, f, min;
procedure TRANSL (A, B, C, D, E, F); value B, C, F; integer B, C, E, F;
integer array A, D; begin ... end TRANSL;
n := 1; h := -1; min := rl + 1;
L1: if scan (subj [rule [n]], a, aux) ≠ 0 then
      begin TRANSL (input, init [n], final [n], output, f, h); h := f - 1;
      comment The segment (of input) being a value of an auxiliary intermediate
      symbol is translated (optionally) and the obtained text is embedded into
      output.;
```

```
L2: d := dist [n] + 1; dist [n] := 0; n := sup [n];
    k := scan (−rule [n] − 1, g, gram) − d
  end ascend to the level sup [n] after exhausting the right-hand side of rule [n]
      or after embedding the translation of the part of input into output
  else k := scan (−rule [n] − 1, g, gram) − 1;

L3: if gram [k] < 0 then goto if n = 1 then L4 else L2;

if scan (gram [k], nt, nonterm) = 0 then
  begin output [h] := gram [k]; h := h − 1; k := k − 1; goto L3 end
  The corresponding terminal symbol from rule [n] is put into output.;

for j := n + 1 step 1 until m do
  if sup [j] = n then begin
                        if if dist [j] ≠ 0 then dist [j] < min else false then
                          begin min := dist [j]; d := j end
                        end The last up to now unused intermediate in rule [n]
                          is found.;

min := rl + 1; n := d; goto L1;

L4: end SYNTHETIZE;

body: STBRL ANALYSIS (gram1, g1, nonterm, nt, term, tr, subj, nr, select, goal,
      input, il, sup, dist, rule, init, final);

WELL TRANSLATE (subj, aux, a, pi1, pi2, per, sup, dist, rule);

SYNTHETIZE (gram2, g2, rl, nonterm, nt, subj, aux, a, output, sup, dist, rule,
init, final);
END:

end GENERALIZED GOOD TRANSLATION;
```

4.

It would be more natural to write analysis and synthesis algorithms in the form of *recursive procedures*. But they were tried out (in the form published here) on the computer DATASAAB D21 and D21 ALGOL does not allow recursive procedures.

The following grammars have been used. Input langauge grammar (a fragment of the ALGOL 60 syntax for arithmetic expressions):

```
⟨letter⟩ ::= b | c | d | e | i | k | l | m | n | p | s
⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨identifier⟩ ::= ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩
⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned integer⟩ ⟨digit⟩
⟨decimal fraction⟩ ::= . ⟨unsigned integer⟩
⟨unsigned number⟩ ::= ⟨unsigned integer⟩ | ⟨decimal fraction⟩ |
                      ⟨unsigned integer⟩ ⟨decimal fraction⟩
```

⟨*adding operator*⟩ ::= + | −
⟨*multiplying operator*⟩ ::= × | /
⟨*primary*⟩ ::= ⟨*identifier*⟩ | ⟨*unsigned number*⟩ | (⟨*arithmetic expression*⟩)
⟨*factor*⟩ ::= ⟨*primary*⟩ | ⟨*factor*⟩ ↑ ⟨*primary*⟩
⟨*term*⟩ ::= ⟨*factor*⟩ | ⟨*term*⟩ ⟨*multiplying operator*⟩ ⟨*factor*⟩
⟨*arithmetic expression*⟩ ::= ⟨*term*⟩ | ⟨*adding operator*⟩ ⟨*term*⟩ |
⟨*arithmetic expression*⟩ ⟨*adding operator*⟩ ⟨*term*⟩

Output language grammer (for arithmetic expression written in reverse Polish notation): All rules with ⟨letter⟩, ..., ⟨multiplying operator⟩ the subjects remain unchanged. Then it follows

⟨*primary*⟩ ::= ⟨*identifier*⟩ | ⟨*unsigned number*⟩ | ⟨*arithmetic expression*⟩
⟨*factor*⟩ ::= ⟨*primary*⟩ | ⟨*primary*⟩, ⟨*factor*⟩ ↑
⟨*term*⟩ ::= ⟨*factor*⟩ | ⟨*factor*⟩, ⟨*term*⟩ ⟨*multiplyng operator*⟩
⟨*arithmetic expression*⟩ ::= ⟨*term*⟩ | ⟨*term*⟩ : ⟨*adding operator*⟩ |
⟨*term*⟩, ⟨*arithmetic expression*⟩ ⟨*adding operator*⟩

If we declare ⟨identifier⟩ and ⟨unsigned number⟩ the auxiliary intermediate symbols, only rules from the 31st one (i.e. ⟨adding operator⟩ ::= +) to the end may be put into gram2. Then arrays *pi*1 and *pi*2 are e.g.

*pi*1: −37, 2, −39, 1, 3, −41, 1, 2, 3, −43, 1, 2, −44, 1, 2, 3, 0
*pi*2: 0, 1, 0, 4, 2, 0, 4, 1, 2, 0, 3, 1, 0, 4, 1, 2, 0

The following declaration of the procedure TRANSL has been used;

procedure *TRANSL* (*A*, *B*, *C*, *D*, *E*, *F*); **value** *B*, *C*, *F*; **integer** *B*, *C*, *E*, *F*;
integer array *A*, *D*;

begin
for *j* := 0 **step** 1 **until** *C* − *B* **do** *D*[*F* − *j*] := *A*[*C* − *j*];
E := *F* − *C* + *B*
end;

Therefore identifiers and unsigned numbers from input are put into output without any change.

Besides others also expression

((*d*21 − *i*1905*c*) ↑ .5 ↑ (*minsk*22 − 1) + (−*ibm*360 + 13.0))/*e*4100 − (*e*803 + 19) × *lps*1

was translated and the correct result

*lps*1, 19, *e*803 + ×, *e*4100, 13.0, *ibm*360 :− +, 1, *minsk*22 −, .5, *i*1905*c*, *d*21 − ↑ ↑ + / −

was obtained. The phrase marker of the given expression has 157 vertices and was printed in the form shown on Table (only several highest levels are shown here).

Table.

LEVEL	*SUP*	*DIST*	*RULE*	*INIT*	*FINAL*
1	0	0	44	1	65
2	1	1	41	52	65
3	2	1	38	62	65
4	3	1	35	62	65
5	4	1	24	62	65
6	5	1	13	65	65
7	5	2	23	62	64
8	7	1	11	64	64
9	7	2	23	62	63
10	9	1	10	63	63
11	9	2	22	62	62

The described way of analysis (syntax directed analysis) is not too effective and should be replaced by other one in case algorithm is really used. The analysis takes up most of the used time. The problem of efficiency of different methods of syntax directed analysis is open as yet in spite of care of it (cf. e.g. references [4] and [5]). But these problems overpass the scope of this article.

(Received February 10th, 1967.)

REFERENCES

[1] Čulík Karel: Well-Translatable Grammars and ALGOL-like Languages. Formal Language Description Langauges for Computer Programming, Proceedings of the IFIP Working Conference. T.B. Steel, Jr. (Editor). North — Holland Publishing Company, Amsterdam 1966, 76—85.

[2] Čulík Karel: Semantics and Translation of Grammars and ALGOL-like Languages. Kybernetika *1* (1965), 1, 47—49.

[3] Kopřiva Jiří: Generalization of Well-Translation of Formal Languages. Kybernetika *2* (1966), 4, 305—313.

[4] Griffiths T. V., Petrick S. R.: On the Relative Efficiencies of Context-Free Grammar Recognizers. CACM *8* (May 1965), 5, 289—300.

[5] Kuno, Susumu: The Augmented Predictive Analyser for Context-Free Languages — Its Relative Efficiency. CACM *9* (Nov. 1966), 11, 810—823.

Realizace algoritmu zobecněného dobrého překládání na počítači

JIŘÍ KOPŘIVA

Ve tvaru (nerekursivní) procedury jazyka ALGOL 60 je zapsán algoritmus pro tzv. zobecněné dobré překládání bezkontextových jazyků (viz [2], [3]). Obsahuje samostatné nezávislé podprocedury pro syntaktickou analýzu, zobecněný dobrý překlad frázového ukazatele a syntézu. Podrobné vysvětlení tohoto algoritmu je podáno v oddílu 2.

RNDr Jiří Kopřiva, CSc., Laboratoř počítacích strojů, Brno, Třída Obránců míru 21.